REMARKS

Status:

Claims 1-8 are pending and stand rejected under 35 U.S.C. §102(e) as being anticipated by the teaching of U. S. Pat. No. 6,633,888 to Kobayashi.

Claims 1-8 are presented for reconsideration as is explained in the discussion that follows.

Analysis:

Applicant strongly disagrees with the interpretation of the Kobayashi teaching asserted in the recent Office Action in support of the rejection of Applicant's claims.

Looking first to Kobayashi's specification, col. 3, lines 35-40 it is stated: " *"components" [which] are self-contained objects that perform a specified function or procedure. Components have a pre-defined interface that conforms to the object model of the underlying language and generally conform to a "component model" associated with the language.*" (Italics used to indicate quotes, bracket deletion for readability)

Kobayashi sees a need to increase the usability of visual builders to create, modify and test components (see col.4, lines 50-57):
"[However,] current visual builders do not allow components to be tested in this efficient manner for several reasons. For example, current visual builders cannot specify at runtime parameters which are often required by methods in the component. Instead, conventional visual builders require objects to actually be built from the class code and then the objects are tested."

Kubayashi further identifies a need in the art at col. 1, lines 61-66:
"Therefore, a need exists for a method and apparatus for compiling and testing newly created component classes within the visual builder interface. A need exists for visually creating, editing and testing composite component classes, including visually editing properties and associating methods among component classes within the composite component class. "

As a solution to expanding the flexibility of visual builders Kobayashi introduces "proxy beans"(col,.9 lines 20-24):
"As previously mentioned, ***component creator 200 functions to convert Java class code and other Java beans into proxy beans*** *which can be combined to form a "bean-based" application by a conventional visual builder."*(bolding added for emphasis)

To implement proxy beans Kobayashi teaches special add-ons:
... *"In accordance with the principles of the invention, visual builder 214 includes a bean visual environment add-on 212, which allows the proxy beans 210 to appear directly on the builder's object palette. The proxy beans can therefore be manipulated in a conventional fashion using builder 214. Since the proxy beans include the parameters of the methods as attributes, the method parameters can be manipulated directly by changing the attributes of the proxy beans in order to generate a bean-based application 216...The bean compiler 208, the bean visual environment add-on 212 and the universal transport API 206, function together* ***to allow Java objects and beans to be manipulated by means of a standard visual builder 214 regardless of whether the objects were originally designed to be used as beans.*** *(Bolding added for emphasis)*

But, here, there seems a conflict with the application of the art to Applicant's claims. If the proxy, as proposed in the Office Action, is the wrapper of Applicant's claims, it should, according to Applicant's claims, mirror the public interface of the object (software test component). But, Kobayashi teaches a proxy that looks different to the visual builder

than the object. Indeed, the gist of the teaching is that the proxy is added to the pallette of the visual builder even though the object, itself, could not be added. In this regard, see bolded text above regarding conversion by component creator 200 of Koyabashi.

In view of this significant deficiency, it is respectfully requested that the rejections of the claims under 35 U.S.C. §102(e) be withdrawn.

Continuing with the analysis, a proxy object supported by the above identified add-ons can act as a bean for purposes of the visual builder. This expands the ability of the visual builder to create composite components including proxy beans that point to code that was otherwise not usable by the visual builder. This leads to the testing taught by Kobayashi for the composite object that is created.

Figure 20 addresses the testing of such a composite (col 9, line 18-29):
*"FIG. 20 discloses a bean testing method which utilizes the aforementioned visual builder. While this method can be a standalone process, it is also representative of an expansion of step 1250 of FIG. 12. The beans to be tested could have been created from a number of different programming means. In any case the bean is processed by the bean compiler 300, discussed earlier in connection with FIG. 3 to create proxy beans for the methods. **The proxy beans are then wired together to create a composite bean** as discussed above. **The composite bean can be tested and then added to a visual builder palette for use in creating objects or other composite beans**".*

"In step 2002, the bean to be tested is loaded into the tester described above using the visual environment add-on, also as discussed above"(col. 9, lines 38-40).

Next, in step 2004, the bean is displayed in the workspace window by selecting it from the palette. At this point the bean can be edited and modified. In step 2006, the bean can be run by choosing

the "Run" item from the tester Mode menu 1110 or by clicking on the Run toolbar button icon in the toolbar 1120. As previously mentioned, when the methods of proxy beans are invoked, they use the universal transport mechanism to invoke the actual component code in order to test the method as set forth in step 2008. Further, as discussed previously, the method parameters of the original bean are exposed by the proxy components created from the methods of that bean. ***Consequently, each method can be tested fully.*** *(col. 9, lines 41-54)*

Next, in step 2010, the bean is checked to ascertain if it is operating properly. ***In some cases the result of the test can be ascertained visually. For example, if the bean responds to an event by running a graphics image, the image should be displayed. In another example, if the bean had bound properties associated with it, like simultaneously changing the background color of objects created from the bean, the bound property could also be tested by selecting and changing the background color of one bean using the bean tester properties panel and then inspecting whether the background color of the other bean with bound properties also changed. In other cases property values must be checked.*** *(Col. 9, lines 54-65)*

But, once again, there is a conflict regarding application of the teaching to Applicant's claims

There is no recording of capture and playback of user interaction with the public user interface, as called for in all of Applicant's claims. The test proposed by Kobayashi is for the user determine by viewing if the right graphic is displayed or if the color is correct. Where is such capture and playback? It appears to be absent from the teaching.

The Kobayashi teaching with its add-ons allows the running of composite objects including proxy beans to occur in the visual builder an aid to testing and a welcome advance, but not the inventive advance claimed by Applicant. Again a serious deficiency, again a request to withdraw the rejection of claims.

For the reasons stated above, it is believed that Applicant's claims clearly identify inventive subject matter over the prior art. Accordingly, early notice that this case is in condition for allowance is earnestly solicited.

Respectfully Submitted,

George E. Grosser
Reg. No. 25,629l

c/o

IBM Corp.
Dept. T81/Bldg. 503 PO Box 12195
Research Triangle Park, NC 27709

(919)968-7847
Fax 919-254-4330
EMAIL: gegch@prodigy.net